Set-5

1. Implement a C program to eliminate left factoring.

Code:

```
#include <stdio.h>
#include <string.h>
#define MAX_RULES 10
#define MAX_LEN 100
void eliminate_left_factoring(char rules[MAX_RULES][MAX_LEN], int rule_count) {
  for (int i = 0; i < rule_count; i++) {
    char non_terminal = rules[i][0];
    char productions[MAX_RULES][MAX_LEN];
    int prod_count = 0;
    // Extract productions
    char *token = strtok(rules[i] + 3, "|");
    while (token) {
      strcpy(productions[prod_count++], token);
      token = strtok(NULL, "|");
    }
    // Find common prefix
    char prefix[MAX_LEN];
    strcpy(prefix, productions[0]);
    for (int j = 1; j < prod_count; j++) {
      int k = 0;
      while (prefix[k] && productions[j][k] && prefix[k] == productions[j][k])
        k++;
      prefix[k] = '\0';
    }
    if (strlen(prefix) == 0) {
```

```
            printf("%c -> %s\n", non_terminal, rules[i] + 3);
            continue;
        }
        // Print new rules
        printf("%c -> %s%c'\n", non_terminal, prefix, non_terminal);
        printf("%c' -> ", non_terminal);
        int first = 1;
        for (int j = 0; j < prod_count; j++) {
            if (strncmp(productions[j], prefix, strlen(prefix)) == 0) {
                if (!first) printf(" | ");
                printf("%s", productions[j] + strlen(prefix));
                first = 0;
            }
        }
        printf(" | ε\n");
    }
}
int main() {
    int rule_count;
    char rules[MAX_RULES][MAX_LEN];

    printf("Enter number of rules: ");
    scanf("%d", &rule_count);
    getchar();
    printf("Enter grammar rules (Format: A -> α | β):\n");
    for (int i = 0; i < rule_count; i++) {
        fgets(rules[i], MAX_LEN, stdin);
        rules[i][strcspn(rules[i], "\n")] = 0; // Remove newline
```

```
    }
    printf("\nGrammar after Left Factoring:\n");
    eliminate_left_factoring(rules, rule_count);
    return 0;
}
```

2. Write a LEX specification counts the number of characters, number of lines & number of words.

Code:

```
%{
#include <stdio.h>
int char_count = 0, line_count = 0, word_count = 0;
%}

%%
// Ignore spaces, tabs, and count words
[ \t]+   ;
\n      { line_count++; char_count++; }

// Count words (identifiers, numbers, etc.)
[A-Za-z0-9_]+ { word_count++; char_count += yyleng; }

// Count other characters
.      { char_count++; }

%%
int main(int argc, char *argv[]) {
    FILE *file;
    if (argc > 1) {
```

```
    file = fopen(argv[1], "r");
    if (!file) {
        printf("Cannot open file %s\n", argv[1]);
        return 1;
    }
    yyin = file;
  }
  yylex();
  printf("Characters: %d\n", char_count);
  printf("Lines: %d\n", line_count);
  printf("Words: %d\n", word_count);
  return 0;
}
```

3. Write a LEX Program to check the email address is valid or not.

Code:

```
%{
int flag=0;
%}
%%
[a-z . 0-9]+@[a-z]+".com"|".in" { flag=1; }
%%
int main()
{
yylex();
if(flag==1)
printf("Accepted");
else
printf("Not Accepted");
```

```
}
int yywrap()
{ return 1;
}
```

4. Write a LEX program to print all the constants in the given C source program file.

Input Source Program: (sample.c)

```
#define PI 3.14
#include<stdio.h> #include<conio.h>
 void main()
{
        int a,b,c = 30;
printf("hello");
}
```

Code:

```
%{
#include <stdio.h>
%}

%%
// Ignore spaces, tabs, and new lines
[ \t]+   ;
\n      ;

// Recognize constants (integer and floating point numbers)
[0-9]+\.?[0-9]* { printf("Constant: %s\n", yytext); }

%%
int main(int argc, char *argv[]) {
```

```
    FILE *file;
    if (argc > 1) {
        file = fopen(argv[1], "r");
        if (!file) {
            printf("Cannot open file %s\n", argv[1]);
            return 1;
        }
        yyin = file;
    }
    yylex();
    return 0;
}
```